What a Novice Wants: Students Using Program Visualization in Distance Programming Course

Osku Kannusmäki, Andrés Moreno, Niko Myller, and Erkki Sutinen
Department of Computer Science, University of Joensuu, P.O. Box 111, FI-80101 Joensuu, Finland

{okannus, amoreno, nmyller, sutinen}@cs.joensuu.fi

1 Introduction

Evaluation is an important part of the development cycle of a new application. Especially, software tools for learning and teaching should be evaluated early on and the evaluation should guide the application's implementation and development.

The Jeliot family is a collection of program and algorithm visualization tools designed for novices learning programming, algorithms, and data structures (Ben-Ari et al., 2002). The latest member of the Jeliot family (see `http://cs.joensuu.fi/jeliot/`) is Jeliot 3 (Moreno et al., 2004). It is a program visualization tool that is based on the automatic animation of Java programs.

During the development of the different versions of Jeliot, evaluation of the actual use of the software has been an important part of the development process (Sutinen et al., 1997; Markkanen et al., 1998; Lattu et al., 2000, 2003; Ben-Bassat Levy et al., 2003). Qualitative descriptions of the use of the tool and its limitations have been fruitful for further development of Jeliot (Ben-Ari et al., 2002).

In this paper, we present some results from a qualitative analysis of the use of Jeliot 3, as well as students' requests and proposals for the development of Jeliot 3. The students were taking the second programming course in the ViSCoS (Sutinen and Torvinen, 2003) program at the University of Joensuu. Qualitative information was gathered from different sources such as forums and emails with comments and exercises. All these documents were later compiled to form a knowledge base from which we extracted our preliminary conclusions.

2 Related Work

Most software visualization tools have been empirically evaluated in order to test their effectiveness in teaching new concepts (Hundhausen et al., 2002). Most of the results summarized in Hundhausen et al. (2002) were obtained from tests done after, and maybe before, using the visualization tools (pre- and post-tests). However, Kehoe et al. (1999) proposed not to depend on the results of tests, but rather on the results of performing certain tasks in a less constrained environment—a homework learning scenario. Empirical evaluations, while providing important information, are difficult to performed in a distant learning course.

Bassil and Keller (2001) use a method similar to ours when evaluating seven software visualization tools. They use a qualitative and quantitative approach to compare those tools. First they try to fit the tools into the taxonomy described by Price et al. (1993); then they collected the users' opinions of those tools. The participants of their study mentioned the benefits of SV tool they were using. They also proposed further development of the tool of their choice, by integrating it with third-party tools. However, their evaluation was of tools used in industry, not those used in education.

3 Jeliot 3

Jeliot 3 visualizes the execution flow of a Java program by showing the current state of the program (e.g., methods, variables, and objects) and animations of expression evaluations and loops. Jeliot 3 evolved from a previous version called Jeliot 2000. The new version was

developed in order to achieve two new goals: to provide support for object-oriented programs, and to improve software modularity. Jeliot 3 still retains the user interface and the animation style from Jeliot 2000.

Currently, Jeliot 3 animates a larger subset of the Java language than Jeliot 2000, with features like values and variables of primitive data types (e.g. int, boolean, char), strings, primitive type 1-dimensional arrays, expressions including operations and assignments, control structures, error reporting and I/O operations. Furthermore, it also animates object-oriented programming (e.g. inheritance and method calls). The implementation of Jeliot 3 now makes it easier to develop new extensions due to its modularity and through the use of program trace code (M-code) that provides the means for communication between the visualization engine and a Java interpreter, DynamicJava (Hillion, 2002).

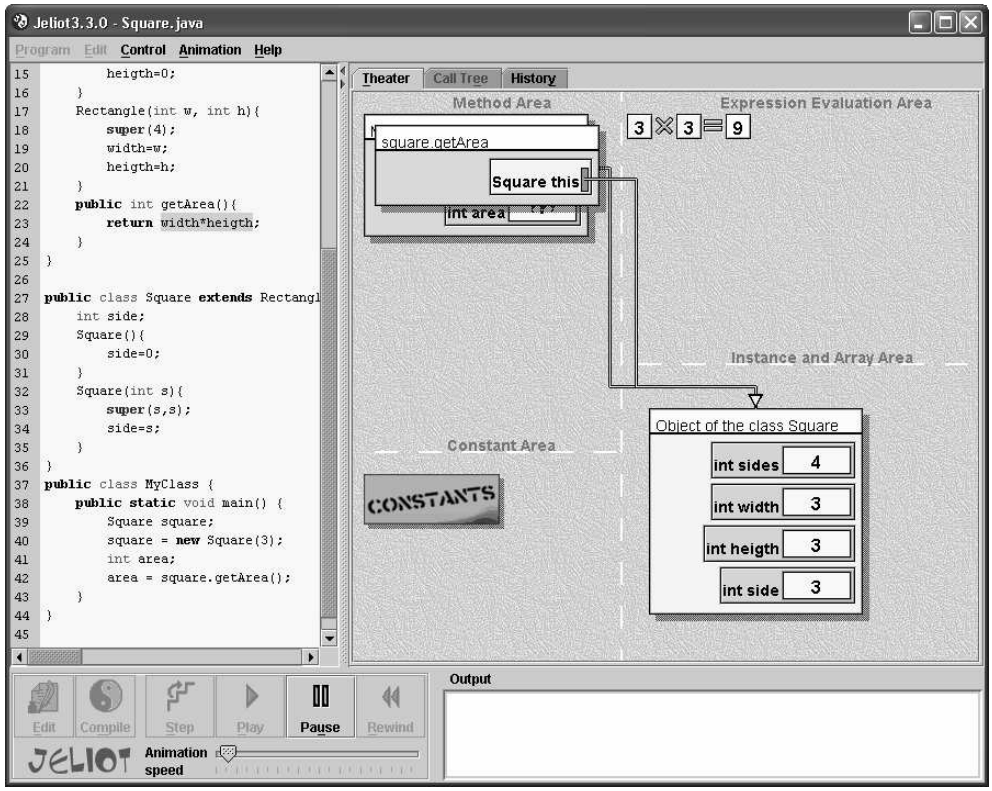

Figure 1: The user interface of Jeliot 3.

The user interface of Jeliot 3 is shown in Figure 1. Interaction takes place in the editor pane which is situated on the left side. There, users modify the source code. By pressing the Compile button, animation begins in the main pane, called the theater, and the editor pane follows the execution of the source code by highlighting the portion of source code being animated. The theater is divided into four sections:

- The *Method area* displays method frames. The frames contain the local variables of the methods.
- The *Expression Evaluation Area* animates the evaluation of the expression. Values are animated from their origin (e.g. a variable value in a method frame) to their place in the evaluation expression area. Complex evaluations are shown as a stack. Input boxes asking for input are laid out here, integrated within the normal flow of the animation.
- Constants appear from the *Constant Area* and are animated like variable values directly to a variable field, or to fill an expression in the expression evaluation area. Furthermore, the static variables are shown in this area.

- Finally, the *Instance and Array Area* displays instances of objects and arrays. They are connected to variables with arrows, indicating the reference semantics of Java. Output data are animated from the theater to the output console at the bottom of the window, where they are shown and stored.

Jeliot 3 can be used in several ways for teaching and learning to program. Here are some examples:

- Lecturers can use Jeliot 3 as a part of the lecture material. They can explain the different concepts of programming through Jeliot animations. This will facilitate the construction by the students of the correct relationship between the animation and the concept, and enable them to apply it later with a reduced possibility of error (Ben-Bassat Levy et al., 2003).
- The students may use Jeliot 3 by themselves after lectures to do assignments.
- Jeliot 3 can be used in an interactive laboratory session, where students may utilize their recently acquired knowledge by writing programs and debugging them through Jeliot 3.
- Finally, Jeliot 3 provides a tool that can aid in courses where external help is not available (e.g. in distance education). Its visualization paradigm creates a reference model that can be used to explain problems by creating a common vocabulary between students and teacher (Ben-Bassat Levy et al., 2003).

4 Empirical Study

As a part of the evaluation study of Jeliot 3, students used Jeliot 3 during their second course of programming—object-oriented programming (3 ECTS points)—in the Virtual Studies of Computer Science (ViSCoS) (Sutinen and Torvinen, 2003) distance learning program at the University of Joensuu's Department of Computer Science. In this study, our aim was to find out how students used the tool and what features they would like to have included in Jeliot in the future.

4.1 Method

In this evaluative case study, we used qualitative methods. With these methods, we tried to determine how students used the tool and what kind of features the novice users of Jeliot 3 would like to have in a visualization and program development tool. The answers to exercises and texts from discussion forum messages from the course were collected and analyzed. Since it was an exploratory study, our aim was not to give precise recommendations about what should be done. Rather the aim of the study was to describe the different ways of using the tool and to analyze the suggestions for improvement given by our sample of programming students.

The texts were analyzed by the third author of this paper who has also been involved in designing and implementing the system. However, this is an exploratory study, the researcher's involvement should not strongly affect the results. During the first reading, the material was coded and the focus of the study was decided. The methodological approach taken was similar to the method Lattu et al. (2003) used in their qualitative study concerning the use of visualization tools as demonstration aids. After the first reading, the coding was revised and the codes were grouped into three main categories. The *usage patterns* category describes the different uses of the tool. The *usage problems* category describes the problems that students encountered while using the tool. Those problems are divided into general, visualization, language, and code-editing related problems categories. The students also described their feelings and opinions, both positive and negative, about using Jeliot 3 as a visualization tool; these descriptions were put in the category *opinions and suggestions*.

4.1.1 Participants

The participants were secondary school students who were taking part in the ViSCoS program. During the course of study, they were taking the second programming course in the ViSCoS program, which dealt with object-oriented programming.

There were 57 students who took part in the second programming course and who agreed to be included in the study. However, only 35 of the students actually returned any assignments that could be analyzed.

Since some of the students were more experienced with computers, especially in programming, than others, and this could have affected the requirements of the program visualization tool for the students. A few students had previous knowledge about Jeliot 2000 and were familiar with its basic features. This is important because the user interface of Jeliot 3 is very similar to Jeliot 2000.

4.1.2 Procedure and Materials

The students used Jeliot 3 in four assignments. They had about one and one half weeks to complete each of the assignments. This means that Jeliot 3 was used, in total, for about six weeks. Each assignment consisted of two questions where students were not required to use Jeliot 3, and two questions where they were required to use Jeliot 3. The assignments were related to the learning material that was divided into chapters, so that each chapter could be completed in about a week. After each assignment, the students were asked to reflect on their usage of Jeliot 3. In addition to the students' reflections, we used the messages on the discussion board, and the feedback emails from the course in the analysis.

The grades of the previous course and this course were used to divide the students into strong ($n = 9$), mediocre ($n = 12$) and weak ($n = 14$). These categories were then used to indicate different levels of requirements as the students had different knowledge levels.

4.2 Results and Discussion

Four different patterns of doing the exercises were found. In two of the patterns, Jeliot 3 was used. In the other two patterns, students only used Jeliot 3 to be sure that their program ran on Jeliot 3, or they did not use it at all.

- Jeliot was used for coding and visually debugging and testing the program.
- A text or code editor was used to code the program and Jeliot was used to visually debug and test the program.
- A text or code editor was used to code the program and a standard Java compiler and JVM were used for debugging and testing.
- An advanced IDE (e.g. Eclipse) was used to code, debug and test the program.

Figure 2 shows how the different patterns appeared in the different user groups. As there were several assignments during the experiment, the same student could report different usage of the tool during different assignments, so the bars can add up to more than 100 per cent; some students did not report anything so some groups do not add up to 100 per cent.

In the first course of programming, students could choose the environment and working style that suited them. During that time they may have got used to a certain environment and working style. When they needed to switch to Jeliot 3, the switch may have indicated changes to their working style, and this could have increased the discomfort with Jeliot 3 and caused the rejection of the tool.

On the other hand, some students found Jeliot to be a nice environment to work with and used it to solve the problems as suggested. Some of the more advanced students used

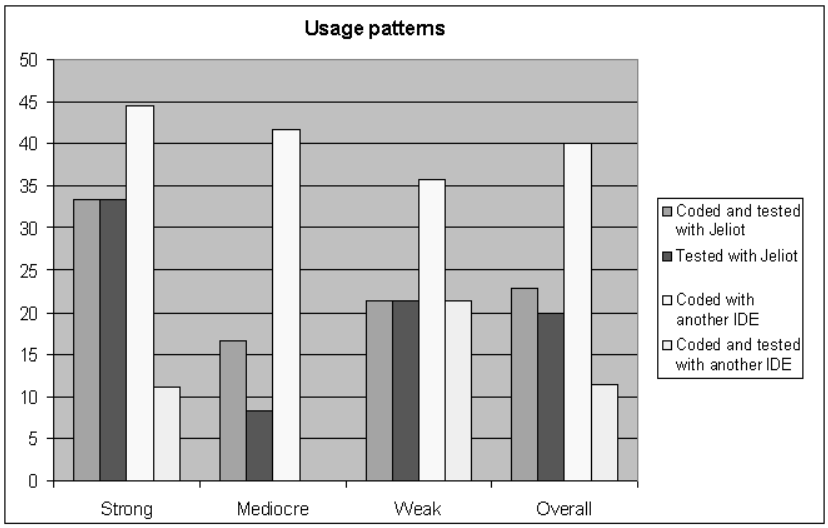

Figure 2: Patterns of use found from the students answers (percent).

advanced IDEs or just a simple text editor, Java compiler, and JVM for development. This indicates that already in the second programming course students' requirements for a learning tool have changed according to their knowledge level.

Different kinds of *usability problems* were identified from the students' answers and comments. They were divided into four different subcategories: *general*, *language*, *editor and visualization related problems*, and *propositions*.

Most of the general usage difficulties had one common feature: they were somehow related to the fact that Jeliot 3 is still under development and there are bugs and unimplemented features that can cause problems. The fact that students needed to install a second version of the tool during the course produced some extra problems. The installation problems of Java SDK also contributed to some of the adoption issues. These problems should be solved as the tool matures.

In several cases, students said that the *error messages were unclear and hard to interpret*. Error messages are an important part of the learning process and they need to be modified according to the user's knowledge (Hristova et al., 2003; Lang, 2002). This indicates that novices would need different kinds of error messages compared to more advanced users. In a similar way, example programs and the user guide of Jeliot were said to be insufficient.

Interestingly, several students seemed to have a misconception about the Java language used by Jeliot 3 since they claimed that *Jeliot 3 was not using standard Java*. However, the subset of Java language that can be visualized with Jeliot 3 is in accordance with Java Language Specification. The subset, however, was different from the one they had been used to, so this may have explained the complaints. The main differences between the Java specification and the subset of Java implemented in Jeliot 3 were: different I/O classes, the Java language support was not full, and the possibility to write `main()` instead of `main(String[] args)` at the beginning of the program (even though `main(String[] args)` is also accepted in Jeliot). The reason for accepting the simpler `main()` method was that it simplifies programs for the novices, so that during the first weeks they do not need to understand parameters and methods.

The students had gotten used to the code editor that they had been using during the first programming course. This seems to have affected the students' requirements for the editor since find & replace and other common editor commands, syntax highlighting, auto-indentation, and bracket indication were requested. The error messages were also requested to be shown during coding as in Eclipse.

Currently, Jeliot 3 requires that all the classes be in a single file, because we want to reduce complexity, and furthermore, code visualization is easier to grasp when the whole program is in a single file. Several students proposed that classes should be in separate files and that it should be possible to run several files at the same time. This is reasonable demand when the programs grow larger as in the second programming course, but before that several files can just confuse the novices.

The problems related to visualization came mostly from those students that used Jeliot 3 only to test the programs, while using an IDE or a code editor and a Java compiler. They complained that the animation was too slow, and they even wanted to have it disabled, or at least to have the possibility to only visualize certain lines of code. Some of the students also said that they can simulate the program in their mind and do not need a visualization tool to do it.

Students also proposed that Jeliot should be used in the first programming course, but not in the second, because it was not needed anymore by that time. The authors of this paper agree with the statement that it should be used in the first course, and that possibly in its current form the tool is not needed in the second programming course by the average student. Nevertheless, it can still help the weak students in the second programming course (see Figure 3). Furthermore, if students would have used Jeliot 3 in the first programming course, it would not have led to a situation where students were not used to the visualization tool and found it hard to change their working style to accommodate a new tool.

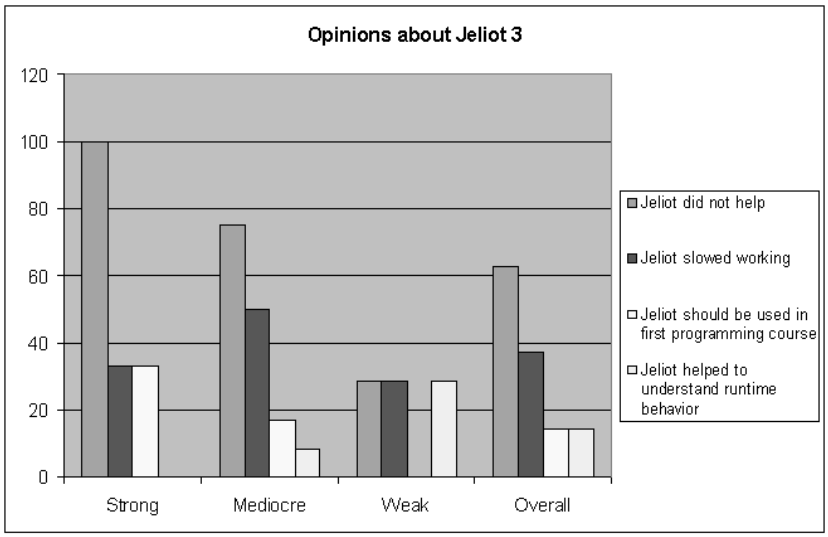

Figure 3: The opinions about Jeliot divided into different knowledge levels (in percentages).

Although students were encouraged to use Jeliot, some students thought that they did not need the tool and regarded it as a waste of time. This is similar to the findings of Ben-Bassat Levy et al. (2003) who found that the strongest students refused to use the previous version of Jeliot, Jeliot 2000. However, in our case, all but the weakest students refused to use Jeliot (see the Figure 3).

One of the students complained that she has a different model of computation that is incompatible with the model that Jeliot uses. The student, however, did not report anything more about her model of computation. If there really are different, but effictive models, developers can program different views or animation paradigms to Jeliot 3 by interpreting the intermediate code generated by Jeliot 3.

In one message, it was mentioned that Jeliot 3 contains too many graphics and that a plain debugger view should also be available. A view that would show classes and their methods and fields was also proposed.

Finally, students also found some positive points concerning when to use Jeliot 3:

- The if-statements and loops were made understandable to the students by Jeliot 3.
- Jeliot 3 helped students grasp objects and their use.
- Jeliot 3 showed in a step-by-step way what happens inside the program, what is wrong with it, and where the errors are.
- The compilation and execution of a Java programming with Jeliot is actually faster although the running time is longer.

Mostly of these comments came from the weak students. The three first comments were anticipated as that is what Jeliot is developed for. The fourth one is also understandable, but the reason for it is that students do not need to compile the programs, as they do when they are using Java compiler from command line. The students need to only push the compile button and the source code is then send to the interpreter which creates the animation.

5 Conclusions

We have described a program animation system called Jeliot 3 and presented some preliminary results from an evaluative study of the use of Jeliot in a distance education context. Students' answers to assignments were qualitatively analyzed and some future research areas were found. The analysis is still preliminary and a deeper analysis is needed. However, it is already clear that in the early stages of learning to program, the needs for programming environments change and that students with different levels of knowledge need different tools.

Advanced students, and even students with just some experience in programming, are very sensitive about changing the tool they have used, unless it provides a significant improvement over the previous tool. This means that the individual characteristics of learners, including levels of knowledge, should be taken into deeper consideration as the students requirements change rapidly.

Jeliot 3 could be improved in several ways. It can become a tool that can help the novices during the first programming course, and then an additional plug-in tool would be used together with an advanced IDE to provide some help for more advanced students. Jeliot 3 is fully automatic and supports novices learning to program, but Jeliot 3 probably could support more advanced students with a semi-automatic view where students could easily modify the properties of the program visualization as is possible in Jeliot I.

In this study, we have identified different kinds of problems that are typically of the problems found by the user of Jeliot 3. The issues can be divided into several categories such as issues related to Jeliot 3's philosophy, its implementation, and programming language considerations. The next step in the research will be to focus on one of these problem areas more carefully. The subsequent analysis could be supported with evaluation that is tightly connected to tool usage, for instance, by logging the users' actions in order to get feedback during the usage of the tool and not just after it.

References

Sarita Bassil and Rudolf K. Keller. A Qualitative and Quantitative Evaluation of Software Visualization Tools. In *Proceedings of the Workshop on Software Visualization*, pages 33–37, Toronto, Canada, May 2001. Held in conjunction with the 23rd Intl. Conf. on Software Engineering (ICSE'2001).

Mordechai Ben-Ari, Niko Myller, Erkki Sutinen, and Jorma Tarhio. Perspectives on Program Animation with Jeliot. In Stephan Diehl, editor, *Software Visualization*, volume 2269 of *Lecture Notes in Computer Science*, pages 31–45. Springer-Verlag, 2002.

Ronit Ben-Bassat Levy, Mordechai Ben-Ari, and Pekka A. Uronen. The Jeliot 2000 rogram Animation System. *Computers & Education*, 40(1):15–21, 2003.

Stéphane Hillion. DynamicJava. WWW-page, 2002. (Koala project) `http://koala.ilog.fr/djava/` (accessed 10.6.2004).

Maria Hristova, Ananya Misra, Megan Rutter, and Rebecca Mercuri. Identifying and Correcting Java Programming Errors for Introductory Computer Science Students. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, pages 153–156. ACM Press, 2003.

Chris D. Hundhausen, Sarah A. Douglas, and John T. Stasko. A Meta-Study of Algorithm Visualization Effectiveness. *Journal of Visual Languages and Computing*, 13(3):259–290, 2002.

Colleen Kehoe, John T. Stasko, and Ashley Taylor. Rethinking the Evaluation of Algorithm Animations as Learning Aids: An Observational Study. Technical Report GIT-GVU-99-10, Graphics, Visualization, and Usability Center, Georgia Institute of Technology, Atlanta, GA, March 1999.

Bob Lang. Teaching new Programmers: a Java Tool Set as a Student Teaching Aid. In *Proceedings of the Inaugural Conference on the Principles and Practice of programming, 2002 and Proceedings of the second workshop on Intermediate Representation Engineering for Virtual Machines 2002*, pages 95–100, National University of Ireland, 2002.

Matti Lattu, Veijo Meisalo, and Jorma Tarhio. A Visualization Tool as a Demonstration Aid. *Computers & Education*, 41(2):133–148, 2003.

Matti Lattu, Jorma Tarhio, and Veijo Meisalo. How a Visualization Tool Can Be Used — Evaluating a Tool in a Research & Revelopment Project. In *12th Workshop of the Psychology of Programming Interest Group*, pages 19–32, Corenza, Italy, 2000. `http://www.ppig.org/papers/12th-lattu.pdf` accessed 10.6.2004).

Janne Markkanen, Pertti Saariluoma, Erkki Sutinen, and Jorma Tarhio. Visualization and Imagery in Teaching Programming. In John Domingue and Paul Mulholland, editors, *10th Annual Meeting of the Psychology of Programming Interest Group*, pages 70–73, Knowledge Media Institute, Open University, Milton Keynes, UK, 1998.

Andrés Moreno, Niko Myller, Erkki Sutinen, and Mordechai Ben-Ari. Visualizing Program with Jeliot 3. In *Proceedings of the International Working Conference on Advanced Visual Interfaces, AVI 2004*, pages 373–380, Gallipoli (Lecce), Italy, 2004.

Blaine A. Price, Ronald M. Baecker, and Ian S. Small. A Principled Taxonomy of Software Visualization. *Journal of Visual Languages and Computing*, 4(3):211–266, 1993.

Erkki Sutinen, Jorma Tarhio, Simo-Pekka Lahtinen, Antti-Pekka Tuovinen, Erkki Rautama, and Veijo Meisalo. Eliot – an Algorithm Animation Environment. Report A-1997-4, Department of Computer Science, University of Helsinki, Helsinki, Finland, 1997. `http://www.cs.helsinki.fi/TR/A-1997/4/A-1997-4.ps.gz` (accessed 10.6.2004).

Erkki Sutinen and Sirpa Torvinen. The Candle Scheme for Creating an on-line Computer Science Program — Experiences and Vision. *Informatics in Education*, 2(1):93–102, January 2003. `http://www.vtex.lt/informatics_in_education/htm/INFE009.htm` (accessed 10.6.2004).